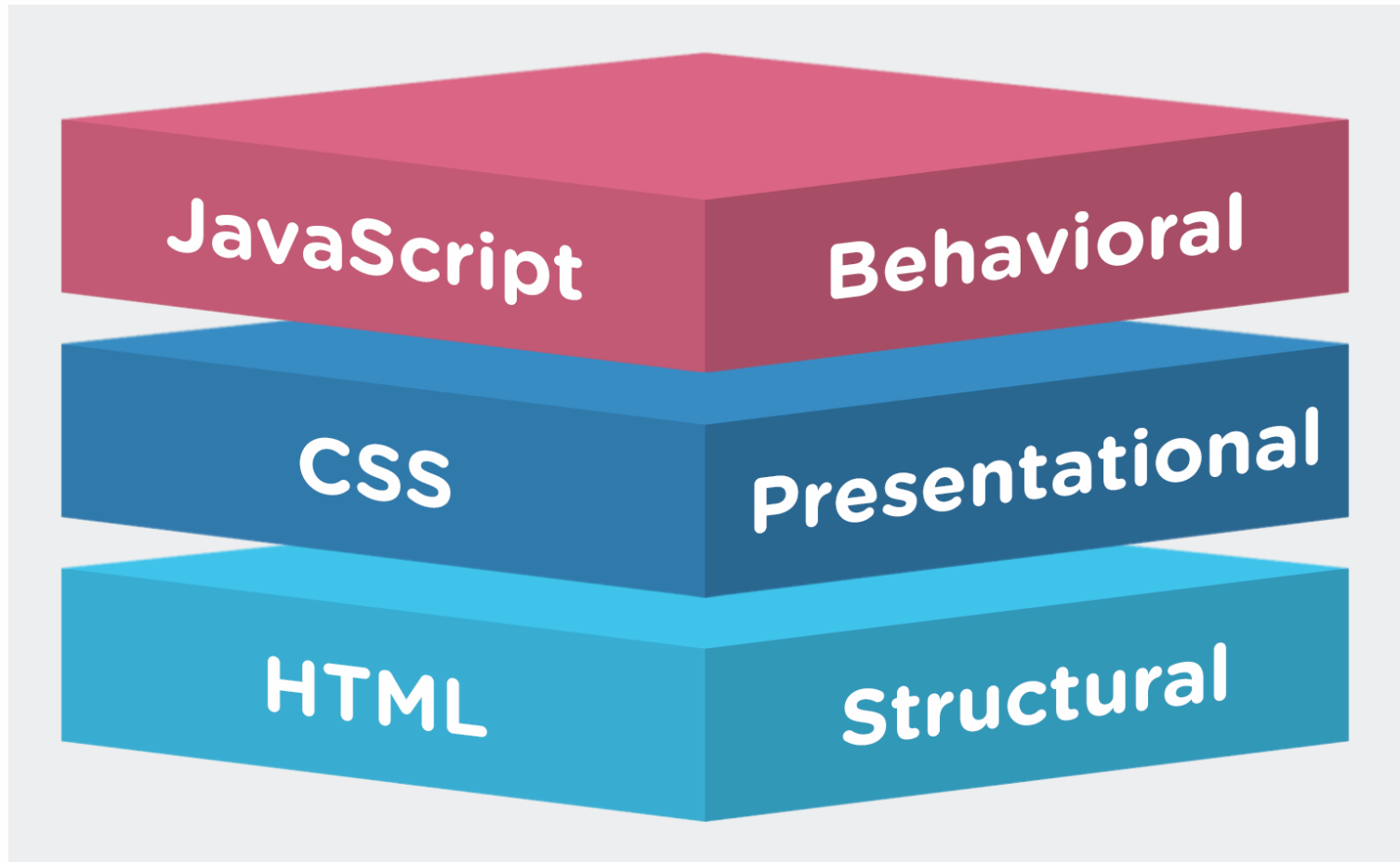


HTML + CSS + JavaScript

Amélioration de l'expérience utilisateur par couches...



Version PDF des slides

Préambule: les langages de description

- Définissent une **structure** et/ou des **règles**
- Sans langage spécifique, la structure n'est pas toujours explicite

Exemple de texte seul:

Polytech Nantes

L'école d'ingénieurs de l'Université de Nantes.

Polytech forme des étudiants ingénieurs, mais pas seulement. Elle délivre aussi des masters pour des publics nationaux et internationaux.

Le détail de ses formations est disponible sur son site web.

Ingénieur informatique: 3 ans d'études pour devenir un expert des technologies numériques.

Master culture numériques: un enseignement interdisciplinaire pour des publics variés.

Préambule: les langages de description (HTML)

- **Structurer** l'information en vue de la **présenter**
 - **Balises**: définissent le sens
 - Exemple: `<h3>Polytech Nantes</h3>`
 - **Attributs**: permettent d'apporter des informations complémentaires
 - Exemple: `site web`

```
<section id="polytech_nantes">
<h3>Polytech Nantes</h3>
<h4>L'école d'ingénieurs de l'Université de Nantes.</h4>
<p>Polytech forme des étudiants ingénieurs, mais pas seulement. Elle délivre
aussi des masters pour des publics nationaux et internationaux.</p>
<p>Le détail de ses formations est disponible sur son <a
href="http://polytech.univ-nantes.fr">site web</a>.</p>
<ul>
  <li><h5>Ingénieur informatique:</h5> 3 ans d'études pour devenir un expert
des technologies numériques.
</li>
  <li><h5>Master culture numériques:</h5> un enseignement interdisciplinaire
pour des publics variés.</li>
</ul>
</section>
```

Préambule: les langages de description (HTML)

Rendu du document HTML dans un navigateur web:

Polytech Nantes

L'école d'ingénieurs de l'Université de Nantes.

Polytech forme des étudiants ingénieurs, mais pas seulement. Elle délivre aussi des masters pour des publics nationaux et internationaux.

Le détail de ses formations est disponible sur son [site web](#).

- **Ingénieur informatique:**

3 ans d'études pour devenir un expert des technologies numériques.

- **Master culture numériques:**

un enseignement interdisciplinaire pour des publics variés.

Préambule: les langages de description (CSS)

- Définir des **règles** de présentation
 - s'appliquent à un/des éléments choisis à l'aide d'un **sélecteur**

```
#polytech_nantes h3, #polytech_nantes h4{
    text-align: center;
}

#polytech_nantes{                /* un sélecteur */
    font-size: 12pt;             /* une règle   */
}

#polytech_nantes li{
    width: 45%;
    display: inline-block;
    vertical-align: top;
    text-align: center;
}
```

Préambule: les langages de description (CSS)

Rendu du même document HTML avec les règles CSS

Polytech Nantes

L'école d'ingénieurs de l'Université de Nantes.

Polytech forme des étudiants ingénieurs, mais pas seulement. Elle délivre aussi des masters pour des publics nationaux et internationaux.

Le détail de ses formations est disponible sur son [site web](#).

Ingénieur informatique:

3 ans d'études pour devenir un expert des technologies numériques.

Master culture numériques:

un enseignement interdisciplinaire pour des publics variés.

Préambule: les langages de description (XML/JSON)

- **Structurer** l'information en vue de la **manipuler**

Departement	Nom	Surface	Habitants
44	Loire-atlantique	6815	1328620
85	Vendée	6720	655506

devient

```
<liste_departements>
  <departement id="44">
    <nom> Loire-Atlantique </nom>
    <surface> 6815 </surface>
    <habitants> 1328620
  </departement>
  <departement id="85">
    <nom> Vendée </nom>
    <surface> 6720 </surface>
    <habitants> 655506
  </departement>
</liste_departements>
```

XML

```
[{ "44": {
  "nom": "Loire-Altantique",
  "surface": "6815",
  "habitants": "1328620"
}},
{ "85": {
  "nom": "Vendée",
  "surface": "6720",
  "habitants": "655506"
}}]
```

JSON

(X)HTML(5) !



Un peu d'histoire...

- 1990 : HTML 1
- 1995 : HTML 2
- 1995 : HTML+ et HTML 3 (non standards)
- 1997 : HTML 3.2 et HTML 4 (standards W3C)
- 1998 : XML 1.0
- 1999 : HTML 4.01 (standards W3C)
- 2000 : XHTML 1.0 (HTML 4 réécrit en XML)
- 2001: XHTML 1.1 (améliorations mineures de XHTML 1.0)
- 2003 : XHTML 2 (concurrent de HTML 5)
- 2008: HTML 5 (premier "Draft" public)
- 2014 : **HTML 5** (standardisé)

Vocabulaire XML

- **Élément** : nom, notion abstraite
 - Exemple : `p`
- **Balise** : forme concrète d'un élément
 - Balise ouvrante : `<element>`
 - Exemple: `<p>`
 - Balise fermante : `</element>`
 - Exemple: `</p>`
- **Attribut** : propriété d'un élément (nom, valeur)
 - `<element nom="valeur" >`
 - Exemple: `<p id="paraPrincipal" >`
- **Commentaire**: non interprétés par les outils XML
 - `<!-- Ceci est un commentaire ! -->`
- **DTD** et **Schema**: définissent le vocabulaire et la grammaire d'un document XML
 - Balises utilisables et leur relations hiérarchiques

Spécificités HTML

- Très proche de XML mais...
- Commence par un doctype plutôt qu'un prologue
 - XML : `<?xml version="1.0"?>`
 - HTML5 : `<!DOCTYPE html>`
 - Mais un document XML peut aussi contenir un doctype !
- Balise auto-fermante `<element/>` ou vide `<element>`
 - Exemple: `` (XHTML) ou `` (HTML5)

Quelques bonnes habitudes (1/2)

- Règles **XHTML** (à suivre aussi en HTML)
 - Les noms des balises doivent être toujours en minuscule
 - Toute balise ouverte doit être fermée
 - Les chevauchements entre balise sont interdits
 - Les noms des attributs doivent être écrits en minuscule
 - Les valeurs des attributs doivent être entre guillemets doubles "..."

Quelques bonnes habitudes (2/2)

- Respecter la sémantique des balises
 - Ex : les tableaux ne servent pas à la mise en page !
- Ne pas abuser des balises génériques (`` et `<div>`)
- Séparer le fond (HTML) de la forme (CSS)
 - Pas de styles en *ligne* dans le code HTML

```
<!-- A éviter ! -->  
<span style="color:red;">Du texte en rouge...</span>
```

- Toujours valider son code HTML (validateur W3C)
- Penser à l'accessibilité (cf. plus loin)

Les différentes version de HTML et leur DOCTYPE

- Un document (X)HTML **doit** commencer par une balise DOCTYPE
 - Il y a de (trop) nombreuses variantes
- Les principales

```
<!-- HTML 5 : la simplicité !-->  
<!DOCTYPE html>
```

```
<!-- HTML 4.01 STRICT (existe aussi en TRANSITIONAL et FRAMESET)-->  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<!-- XHTML 1.1 (STRICT par défaut)-->  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Pourquoi HTML 5 ?

- C'est le dernier standard en date...
- HTML 4 en **plus simple** !
- Meilleure **sémantique** : nouvelles balises `header` , `footer` , `nav` , etc.
- Meilleur **référencement** : microdonnées = nouveaux attributs `itemscope` , `itemtype` , etc.
- Scripting plus simple : **JavaScript** est le langage **par défaut**
- Des formulaires **plus complets** : types `datetime` , `number` , `search` , etc.
- **Audio et vidéo** supportés nativement (plus besoin de plugin)

HTML 5 et JavaScript

- Une **API** importante fait partie de HTML 5
 - Dessin 2D (Canvas)
 - DOM amélioré
 - Stockage local (permanent ou temporaire)
 - Mode hors ligne
 - Communications bi-directionnelles avec un serveur (WebSockets)
 - Géolocalisation
 - Accès à l'historique Web
 - Le drag and drop
 - "Multithreading" (WebWorkers)
 - Communication entre pages (Messaging)
 - Etc.

Un document HTML5 minimal

```
<!DOCTYPE html>      <!-- déclaration du doctype HTML5 -->
<html lang="fr">      <!-- il faut préciser la langue du document html -->
  <head>              <!-- entête: titre, feuilles de style, scripts, etc. -->
    <meta charset="utf-8"> <!-- encodage des caractères : obligatoire ! -->
    <title>title</title> <!-- titre de la page: obligatoire -->
    <link rel="stylesheet" href="style.css"> <!-- feuille de style: optionnel -->
    <script src="script.js"></script>
  </head>
  <body>
    <!-- contenu de la page -->
  </body>
</html>
```

Structure d'un document HTML5 (1/4)

DOCTYPE

- Déjà vu précédemment...
 - Définit le **type de document** HTML
 - Précise la DTD à utiliser
 - Important pour que le navigateur interprète correctement la page

Structure d'un document HTML5 (2/4)

La balise `<html></html>`

- Indique au navigateur qu'il s'agit d'un **document html**
- C'est la **racine** du document html
 - Contient toutes les autres balises

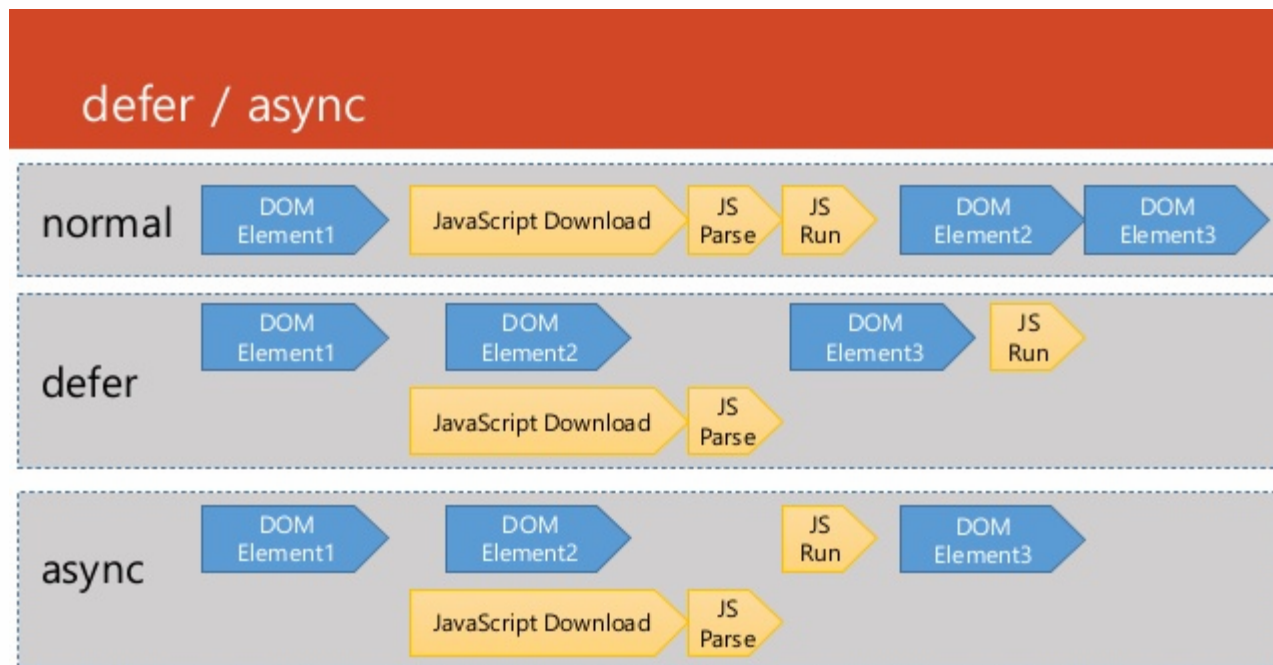
Structure d'un document HTML5 (3/4)

La balise `<head></head>`

- Contient des **(méta) informations** concernant le document
- Balises filles acceptées:
 - `title` : titre du document (**Obligatoire**)
 - `style` : définition de styles intégrée au document
 - `base` : URL de base pour tous les liens relatifs
 - `link` : liens vers des feuilles de style
 - `meta` : méta-données (mots clés, auteur, etc.)
 - `script` : scripts (par défaut JavaScript)
 - Attributs `async` et `defer` pour un chargement asynchrone
 - On peut aussi charger les scripts en fin de `<body>` (obsolète)
 - `noscript` : contenu à insérer si pas de support des scripts

Petite parenthèse...

Chargement des scripts



Structure d'un document HTML5 (4/4)

La balise `<body></body>`

- Le contenu de la page web
- Toutes les balises affichées par le navigateur sont des balises filles de `body`

Les balises d'HTML5

ATTENTION : Liste et description non exhaustive

Ce cours n'a pas vocation à être une référence. D'autres sources d'information sont largement disponibles sur le web.

- <https://developer.mozilla.org/fr/docs/Web/HTML/Reference>
- <https://devdocs.io/html> (agrégateur de documentations)
- Et bien sûr les textes des **standards du W3C** !

Les deux grands types de balises

Bloc

- Contient
 - Du texte
 - Des éléments "En ligne"
 - **Des éléments "Bloc"**
 - Des éléments auto-fermants
- Position
 - **Les uns sous les autres**
- Largeur
 - **Celle du conteneur**
- Exemple: `<div>`, `<p>`, `<h1>`

En ligne

- Contient
 - Du texte
 - Des éléments "En ligne"
 - **Des éléments "En ligne" !**
 - Des éléments auto-fermants
- Position
 - **Les uns à coté des autres**
- Largeur
 - **Sans dimension**
- Exemple: ``, `<a>`, ``

* Il y en a d'autres (list-item, table, table-cell, etc.)

Les deux grands types de balises (exemple)

Bloc

```
<div style="background:yellow;">
  Une boîte jaune
</div>

<div style="background:green;">
  Une boîte verte
</div>
```

Résultat:

Une boîte jaune
Une boîte verte

En ligne

```
<span style="background:yellow;">
  Une boîte jaune
</span>

<span style="background:green;">
  Une boîte verte
</span>
```

Résultat:

Une boîte jaune Une boîte verte

Balises génériques (1/2)

- `<div>` : balise **bloc**
 - Permet de regrouper des éléments
 - Aide à l'agencement du contenu
 - Ne pas en abuser !!!
- `` : balise **en ligne**
 - Permet d'associer un style à une partie de texte
- Ne sont utiles qu'en association avec les **attributs**
 - `class` : permet d'associer un même style CSS à différentes balises
 - `id` : identifie de manière unique une balise
 - `style` : permet d'ajouter du code CSS dans une balise HTML (**à éviter**)

Balises génériques (2/2)

- Exemple:

HTML

```
<div id="div1" class="uneDiv">
  La div 1
</div>

<div id="div2" class="uneDiv">
  La div 2
</div>
```

Résultat sans CSS:

La div 1
La div 2

CSS

```
.uneDiv { display: inline;
          border-style: solid;
          border-color: black; }
#div1 { background-color: green;
         margin: 5px;}
#div2 { background-color: yellow;
         padding: 5px;}}
```

Résultat avec CSS:

La div 1 La div 2

Les attributs globaux (1/2)

Peuvent être définis sur n'importe quel élément HTML

- `accesskey` : raccourci clavier de l'élément
- `class` : permet d'associer un même style CSS à différentes balises
 - classes multiples possibles. Ex: `<div class="classe1 classe2"></div>`
- `contenteditable` : rend l'élément modifiable
- `contextmenu` : permet de référencer un élément `<menu>` (contenant des `<menuitem>`) qui apparaîtra au clic droit sur l'élément
- `data-*` : permet d'intégrer des données au code HTML, qui pourront être accédées via le DOM
- `dir` : direction du texte de l'élément

Les attributs globaux (2/2)

Peuvent être définis sur n'importe quel élément HTML

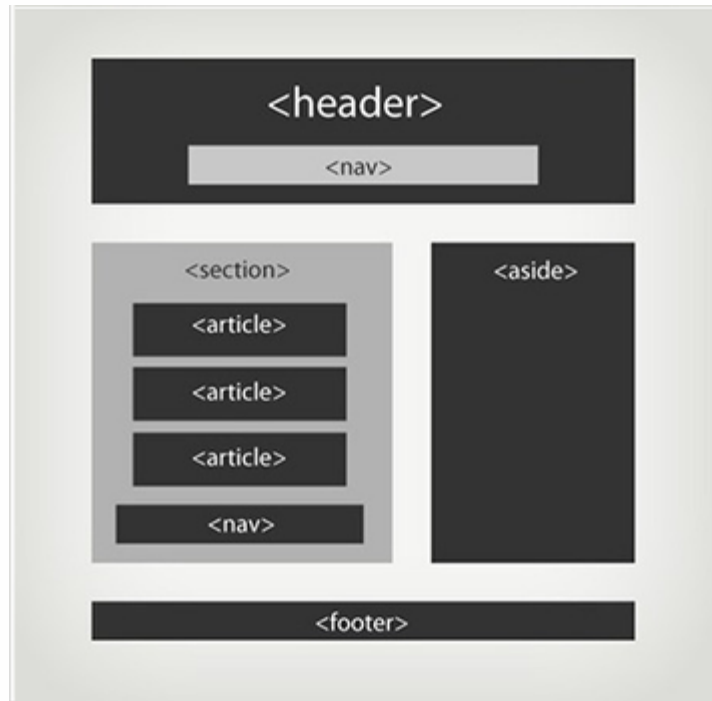
- `hidden` : permet de "cacher" un élément (**pas que graphiquement**)
- `id` : identifiant de l'élément
- `lang` : définit le langage de l'élément
- `style` : permet d'ajouter du code CSS dans une balise HTML (**à éviter**)
- `tabindex` : spécifie l'ordre de navigation de l'élément via la touche "tab"
- `title` : informations complémentaires concernant l'élément, apparaissant le plus souvent sous forme de bulle d'aide
- `translate` : indique si l'élément doit être traduit en cas de traduction de la page

Sections du document

- `<header>` : Section d'**introduction** d'un article, d'une autre section ou du document entier
- `<nav>` : Liens de **navigation** principaux (au sein du document ou vers d'autres pages)
- `<main>` : Le contenu **principal** de la page
- `<footer>` : Section de **conclusion** d'une section ou d'un article, voire du document entier
- `<section>` : Section générique regroupant un **même sujet**, une **même fonctionnalité**, possède un entête ou un titre
- `<article>` : Section de contenu **indépendante**, pouvant être extraite individuellement du document ou syndiquée (flux RSS ou équivalent), sans pénaliser sa compréhension
- `<aside>` : Section dont le contenu est un **complément** par rapport à ce qui l'entoure

* Ce sont toutes des balises de type bloc

Exemple de sections d'un document



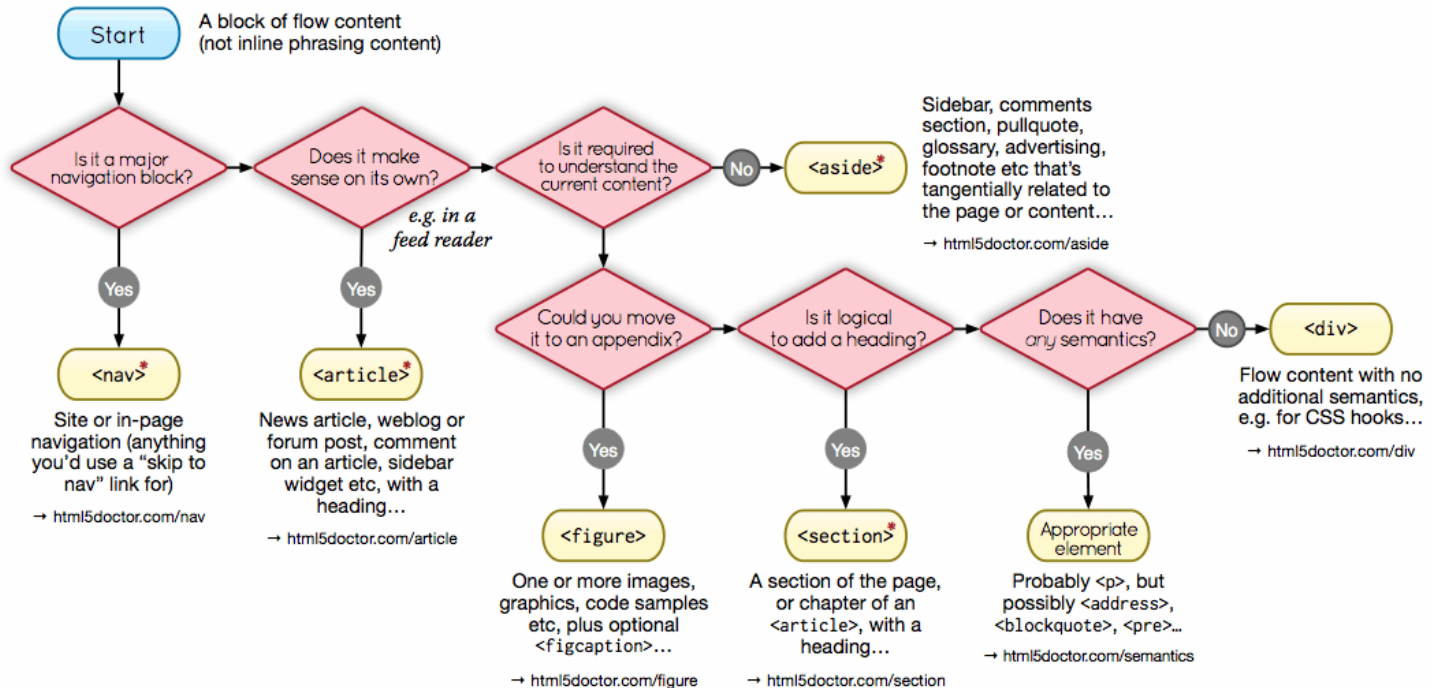
* d'autres solutions sont possibles...

Bien utiliser les éléments de section



HTML5 Element Flowchart Sectioning content elements and friends

By @riddle & @boblet
www.html5doctor.com



* Sectioning content element

These four elements (and their headings) are used by HTML5's outlining algorithm to make the document's outline
→ html5doctor.com/outline

2011-07-22 v1.5
For more information:
www.html5doctor.com/semantics

Les balise de structuration (1/4)

- `<abbr>` : Abréviation
- `<blockquote>` :
Citation (longue)
- `<cite>` : *Citation du titre d'une œuvre ou d'un évènement*
- `<q>` : "Citation (courte)"
- `<sup>` : Texte^{Exposant}
- `<sub>` : Texte_{Indice}
- `` : **Mise en valeur forte**
- `` : *Mise en valeur normale*
- `<mark>` : Mise en valeur visuelle

En **rouge, gras**: les balises de type bloc

Les balise de structuration (2/4)

- `<h1>` :

Titre de niveau 1

- `<h2>` :

Titre de niveau 2

- `<h3>` :

Titre de niveau 3

- `<h4>` :

Titre de niveau 4

- `<h5>` :

Titre de niveau 5

- `<h6>` :

Titre de niveau 6

Les balise de structuration (3/4)

- `<figure>` :

Figure (image, code, etc.)

- `<figcaption>` :

Description de la figure

- `<a>` : **Lien hypertexte**. L'attribut `href` définit l'URL cible

- `
` : **(ne devrait pas servir souvent !)**


Retour à la ligne

- `<p>` :

Paragraphe

- `<hr>` : Ligne de séparation horizontale
-

Les balise de structuration (4/4)

- **<address>** :
Adresse de contact
- **** : ~~Texte supprimé~~
- **<ins>** : Texte inséré
- **<dfn>** : *Définition*
- **<kbd>** : Saisie clavier
- **<pre>** :
Affichage formaté (pour les codes sources)
- **<progress>** : Barre de progression 
- **<time>** : Date ou heure. Ex: Mars 2015

Les listes

- **Liste ordonnée** ``:

```
<ol>
  <li>Premier élément</li>
  <li>Second élément</li>
</ol>
```

Résultat:

1. Premier élément
2. Second élément

- **Liste non ordonnée** ``:

```
<ul>
  <li>Un élément</li>
  <li>Un autre élément</li>
</ul>
```

Résultat:

- Un élément
- Un autre élément

Dans tous les cas les listes contiennent des **éléments** `` et peuvent être imbriquées les unes dans les autres.

Les listes de définitions (plus rares)

- Trois types de balises nécessaires
- **<dl>** : la liste de définitions
 - **<dt>** : un terme à définir
 - **<dd>** : la définition du terme
- Exemple:

```
<dl>  
  <dt>HTML</dt><dd>Langage de représentation de pages web</dd>  
  <dt>CSS</dt><dd>Langage de mise en forme de pages web</dd>  
</dl>
```

Résultat

HTML

Langage de représentation de pages web

CSS

Langage de mise en forme de pages web

Les tableaux

- `<table>` : Balise '**racine**' du tableau. Contient tous les autres éléments
- `<caption>` : **Titre** du tableau (*Optionnel*)
- `<thead>` : **En-tête** du tableau (*Optionnel*)
- `<tbody>` : **Corps** du tableau (*Optionnel*)
- `<tfoot>` : **Pied** du tableau (*Optionnel*)
- `<tr>` : Une **ligne** du tableau
- `<td>` : Une **colonne** du tableau
- Les attributs `colspan` et `rowspan` servent à **fusionner** des cellules

Les tableaux (exemple)

```
<table>
  <caption>Liste
d'étudiants</caption>
  <!-- En-tête (optionnel) -->
  <thead>
    <tr>
      <th>Nom</th>
      <th>Numéro</th>
    </tr>
  </thead>
  <!-- Corps du tableau -->
  <tbody>
    <tr>
      <td>James</td>
      <td>007</td>
    </tr>
    <tr>
      <td>Ulla</td>
      <td>3615</td>
    </tr>
  </tbody>
</table>
```

Résultat

Liste d'étudiants

Nom	Numéro
James	007
Ulla	3615

Les images

- `` : Une **image** dont le fichier source est indiqué par l'attribut `src`.
 - L'attribut `alt` fournit une description textuelle de l'image (accessibilité).
 - L'attribut `srcset` permet de définir différentes tailles d'image (responsive + high dpi)
 - Peut être insérée dans une balise `<a>` pour créer une image cliquable
- `<picture>` et `<source>` : Solution HTML5, alternative à `srcset`, pour spécifier **différentes tailles d'images** (responsive + high dpi) ou **différents formats** (ex: JPG et WebP)
- `<canvas>` : **Surface** (matricielle) de dessin, qui sera manipulée via l'API javascript éponyme.
 - A besoin d'un attribut `id` pour être référencée en JavaScript
 - Ne pas oublier de lui donner une largeur et une hauteur (de préférence en CSS)

Les images (exemples)

- `` et `srcset`

```

```

- `<picture>` et `<source>`

```
<picture>  
  <source media="(min-width: 650px)" srcset="images/kitten-large.png">  
  <source media="(min-width: 465px)" srcset="images/kitten-medium.png">  
  <!-- un élément <img>, pour les navigateurs ne supportant pas <picture> -->  
    
</picture>
```

- `<canvas>`

```
<canvas id="drawing_area" width="800" height="600"></canvas>
```

Les balises Audio

- Nouveau depuis HTML5 : avant, aucun moyen standard de jouer du son
- `<audio>` : Définit un **son** à jouer sur une page
- Attributs utiles:
 - `src` : L'URL du son à jouer
 - **Attention** : tous les navigateurs ne supportent pas les mêmes codecs. Il faut au moins fournir le son au formats Ogg/Vorbis et MP4/AAC
 - `autoplay` : Comme son nom l'indique...
 - `controls` : Permet d'afficher une interface de lecture du son
 - `type` : Le type MIME du son à jouer

Exemple:

```
<audio controls src="http://freedownloads.last.fm/download/604040204/Goliath.mp3"
type="audio/mp3">
Description alternative ici !
</audio>
```

Les balises Vidéo

- **Nouveau depuis HTML5** : avant, aucun moyen standard de jouer des vidéos
- `<video>` : Définit une **vidéo** à jouer sur une page Attributs utiles:
 - `src` : L'URL du son à jouer
 - **Attention** : comme pour l'audio, tous les navigateurs ne supportent pas les mêmes codecs. Le .h264 (mp4) est **de mieux en mieux supporté**, mais il faut aussi compter avec WebM et Theora.
 - `autoplay` : Comme son nom l'indique...
 - `controls` : Permet d'afficher une interface de lecture du son
 - `type` : Le type MIME du son à jouer

Exemple:

```
<video controls="controls">  
  <source src="video.mp4" type="video/mp4" >  
  <source src="video.webm" type="video/webm" >  
  <source src="video.ogv" type="video/ogg" >  
  Description alternative ici !  
</video>
```

Les iFrame

- `<iframe>` : Permet d'inclure une page web à l'intérieur d'une autre (ex: une carte Google Map).
- Attributs utiles:
 - `src` : URL du contenu web à intégrer
 - `srcdoc` : Code HTML à intégrer (alternative à `src`)
 - `sandbox` .blue[(HTML5)]: Permet de spécifier certaines options de sécurité
 - `allowfullscreen` : Permet au contenu de l'iFrame de remplir tout l'écran

Exemple basique:

```
<iframe id="theFrame" src="http://example.com/"></iframe>
```

Exemple de vidéo YouTube:

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/kPHLewY60c0"
frameborder="0" allowfullscreen></iframe>
```

Autre contenu

- `<embed>` (HTML5) : La bonne manière d'intégrer du contenu externe via un plugin (Ex: Flash)
 - Attributs utiles:
 - `src` : URL du contenu externe à intégrer
 - `type` : Type MIME du contenu externe
 - Pas de possibilité de paramétrage supplémentaire
- `<object>` : Element plus générique, représentant du contenu externe
 - Attributs utiles:
 - `data` : URL du contenu externe à intégrer
 - `type` : Type MIME du contenu externe
 - On peut passer des paramètres supplémentaire a l'objet externe via via l'inclusion de balises `<param>` et ses attributs `name` et `value`

Autre contenu (exemples)

- `<embed>` (HTML5)

```
<!-- Une vidéo lue avec le plugin Quicktime -->  
<embed type="video/quicktime" src="movie.mov" width="640" height="480">  
  
<!-- Un contenu (ex: animation) Flash -->  
<embed type="application/x-shockwave-flash" src="anim.swf" width="640"  
height="480">
```

- `<object>`

```
<!-- Un contenu (ex: animation) Flash -->  
<object data="anim.swf" type="application/x-shockwave-flash"></object>  
  
<!-- Un contenu (ex: animation) Flash avec paramètres-->  
<object data="anim.swf" type="application/x-shockwave-flash">  
  <param name="foo" value="bar">  
</object>
```


Les formulaires

- C'est en grande partie grâce à eux que l'on peut **interagir avec l'utilisateur** et **communiquer avec le serveur web** (hors AJAX)
- La balise `<form>` permet de **déclarer un formulaire**
- Attributs utiles
 - `action` : URL de la page / script serveur qui traitera les données du formulaire
- `method` : Méthode HTTP à utiliser (**GET** ou POST)
- `enctype` : Seulement utilisé si `method=post`
 - `application/x-www-form-urlencoded` : encodage des données en *percent encoding*, comme les URI
 - `multipart/form-data` : Pas d'altération des données. Nécessaire en cas d'envoi de fichier (`<input type="file">`)
 - `text/plain` : Seuls les espaces sont convertis en '+'

Regroupements et labels

- `<fieldset>` : Permet de créer des **groupes de widgets** `<input>`
 - Est particulièrement important pour les boutons 'radio'
- `<legend>` : **Légende** du groupe créé avec un `<fieldset>`
- `<label>` : Permet d'associer une **étiquette texte** à un champ `<input>`
 - L'attribut `for` indique l'`id` du champ à étiqueter
 - Important pour que votre formulaire soit compatible avec les outils d'**accessibilité**
- `<output>` : Permet de créer un champ qui recevra le **résultat d'un calcul** issu de plusieurs autres champs
 - L'attribut `for` spécifie une liste d'`id` de champs, séparés par des espaces

Exemple de regroupement

```
<form>
  <fieldset>
    <legend>Taille de boisson</legend>
    <p>
      <input type="radio" name="size" id="taille_1"
value="petit" >
      <label for="taille_1">Petit</label>
    </p>
    <p>
      <input type="radio" name="size" id="taille_2"
value="moyen" >
      <label for="taille_2">Moyen</label>
    </p>
    <p>
      <input type="radio" name="size" id="taille_3"
value="grand" >>
      <label for="taille_3">Grand</label>
    </p>
  </fieldset>
</form>
```

Résultat:

Taille de boisson

- ☐ Petit
- ☐ Moyen
- ☐ Grand

Exemple labels

```
<form>
  <p>
    <input type="checkbox" id="j_aime_1" name="aime_pommes" value="1">
    <label for="j_aime_1">J'aime les pommes</label>
  </p>
  <p>
    <label for="j_aime_2">J'adore les sushis</label>
    <input type="checkbox" id="j_aime_2" name="aime_sushis" value="1">
  </p>
</form>
```

Résultat:

- ☐ J'aime les pommes
- ☐ J'adore les sushis

Exemple de champ `<output>`

```
<!-- notre machine à multiplier -->
<form oninput="output.value = (val1.valueAsNumber || 0) * (val2.valueAsNumber || 0)" id=foo>
  <input type=number name=val1> X
  <input type=number name=val2> =
  <output name=output for="val1 val2" form=foo>0</output>
</form>
```

Résultat:

X = 0

Les widgets

- L'élément `<input>` permet de définir de **nombreux types de widget**
 - Il est généralement déclaré à l'intérieur d'un paragraphe `<p>`
- L'attribut `type` spécifie le **type du widget**
 - Champs textes mono-ligne: `text`, `email`, `password`, `search`, `tel`, `url`
 - Champs non textuels: `checkbox`, `color`, `file`, `hidden`, `number`, `radio`, `range`
 - Date et heure: `date`, `datetime`, `datetime-local`, `month`, `time`, `week`
 - Boutons: `button`, `image`, `reset`, `submit`
- Nombreux autres attributs spécifiques à chaque type d'input (ex: `min`, `max`, `list`, etc.) nécessaires pour l'**auto-validation** des formulaires

Les boutons

- L'élément `<button>` est très proche d'un champ `<input>` de type bouton
- Contrairement à `<input>`, il peut **contenir du texte formaté ou des images**
- Trois types de boutons (attribut `type`)
 - `submit`: Envoie les données du formulaire au serveur
 - `reset`: Réinitialiser tous les widgets du formulaire à leur valeur par défaut
 - `anonymous`: Pas de comportement pré-défini. Doit être exploité à l'aide de code JavaScript

Les widgets (exemple)

```
<form>
  <p>
    <label for="t1">Texte:</label>
    <input id="t1" type="text" required pattern="^test1|test2$">
  </p>
  <!-- Champ email pouvant contenir plusieurs adresse -->
  <p>
    <label for="e1">Email:</label>
    <input id="e1" type="email" required multiple>
  </p>
  <!-- Champ mot de passe (n'affiche pas le mot de passe) -->
  <p>
    <label for="p1">Mot de passe:</label>
    <input id="p1" required type="password">
  </p>
  <!-- Champ de recherche avec historique (même site) et placeholder -->
  <p>
    <label for="s1">Recherche:</label>
    <input id="s1" type="search" placeholder="Faites une recherche !">
  </p>
  <!-- le bouton de validation -->
  <button type="submit">Tester la <strong>validation</strong></button>
</form>
```


Les widgets (exemple)

Resultat:

Texte:

Email:

Mot de passe:

Recherche:

Tester la **validation**

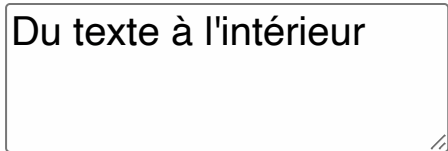
Zones de texte multilignes

- L'élément `<textarea>` peut contenir plusieurs lignes de texte
- On définit son nombre de lignes et colonnes avec les attributs `rows` et `cols`

Exemple:

```
<textarea cols="20" rows="3">Du texte à l'intérieur</textarea>
```

Resultat:



Du texte à l'intérieur

Listes déroulantes

- L'élément `<select>` définit une liste déroulante
 - `<option>` : permet de spécifier les valeurs possibles
 - L'attribut `selected` de `<option>` indique la valeur par défaut de la liste
 - `<optgroup>` : Permet de regrouper ces valeurs
 - L'attribut `multiple` de `<select>` permet de sélectionner plusieurs options

Exemple:

```
<select>
  <optgroup label="Fruits">
    <option>Banane</option>
    <option selected>Pomme</option>
    <option>Orange</option>
  </optgroup>
  <optgroup label="Légumes">
    <option>Carotte</option>
    <option>Pomme de terre</option>
    <option>Choux</option>
  </optgroup>
</select>
```

Resultat:

Pomme ▼

Auto-complétion

- L'élément `<datalist>` définit une liste de choix possibles pour un champ texte
 - La liste est reliée au champ texte via l'attribut `list` du champ texte et l'`id` de la liste
 - `<option>` : Permet de spécifier les valeurs possibles

Exemple:

```
<label for="inpChocType">Chocolats:
</label>
<input type="text" id="inpChocType"
list="chocType">
<datalist id="chocType">
  <option value="blanc" >
  <option value="lait" >
  <option value="noir" >
</datalist>
```

Resultat:

Chocolats

Progression (1/2)

- `<meter>` : Représente une **valeur fixe**, comprise entre un `min` et un `max`

Exemple:

```
<p>Espace utilisé dans votre webmail :  
  <meter low="1000" high="6500" max="7500" value="6985">6 985 Mo</meter>  
</p>
```

Résultat:

Espace utilisé dans votre webmail : 

Progression (2/2)

- `<progress>` : Représente une valeur qui peut **varier au cours du temps** entre un `min` et un `max`

Exemple:

```
<p>Avancement de la tâche à effectuer :  
  <progress id="avancement" value="50" max="100"></progress>  
  <span id="pourcentage"></span>  
  <input type="button" onclick="modif(-10);" value="-">  
  <input type="button" onclick="modif(10);" value="+>  
</p>
```

Résultat:

Avancement de la tâche à effectuer : 

Web sémantique : les microdonnées

- Ajout d'attributs aux tags d'une page HTML pour améliorer l'**indexation des pages** par des moteurs de recherche et le **partage de données**
- Cinq nouveaux attributs
 - `itemscope` : Crée un élément et indique que les descendants de cette balise HTML contiennent des informations à son sujet
 - `itemtype` : Une URL pointant vers un vocabulaire qui décrit l'élément et ses propriétés (cf. **schema.org**)
 - `itemid` : Un identifiant unique pour l'élément
 - `itemprop` : Indique que la balise contient la valeur de la propriété indiquée. Les noms des propriétés et leurs valeurs possibles sont indiquées dans le vocabulaire
 - `itemref` : Permet d'associer une balise non-descendante à une balise avec l'attribut `itemscope`

Microdonnées : exemple

```
<p>
  Bonjour, je m'appelle Robert De Niro. Je suis un étudiant ingénieur à Polytech
  Nantes.
  Mes amis m'appellent Bob. Vous pouvez visiter ma page web ici :
  <a href="fr.wikipedia.org/wiki/Robert_De_Niro">fr.wikipedia.org</a> .
  J'habite Rue Christian Pauc, Nantes, Loire Atlantique.
</p>
```

Devient (avec les microdonnées):

```
<p itemscope itemtype="http://schema.org/Person">
  Bonjour, je m'appelle <span itemprop="name">Robert De Niro</span>.
  Je suis un <span itemprop="jobTitle">étudiant ingénieur</span> à
  <span itemprop="affiliation">Polytech Nantes</span>.
  Mes amis m'appellent <span itemprop="additionalName">Bob</span>.
  Vous pouvez visiter ma page web ici :
  <a href="http://fr.wikipedia.org/wiki/Robert_De_Niro"
  itemprop="url">fr.wikipedia.org</a>.
  <span itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
    J'habite
    <span itemprop="streetAddress">Rue Christian Pauc</span>,
    <span itemprop="addressLocality">Nantes</span>,
    <span itemprop="addressRegion">Loire Atlantique</span>.
  </span>
</p>
```


Accessibilité

- Quelques règles à suivre pour rendre ses pages plus accessibles
 - Toujours utiliser l'**attribut alt** lorsque l'on utilise des images
 - Faire attention à l'**ordre des informations** (la page doit être compréhensible sans le CSS)
 - Ne pas utiliser d'élément `` pour représenter du texte
 - Utiliser CSS pour changer la police ou remplacer le texte par une image
 - Mais utiliser `` (et `alt`) pour les images porteuses d'information
 - Respecter et exploiter au maximum la **sémantique** des balises
 - Utiliser les **attributs ARIA** quand c'est nécessaire
 - Seulement pour les rôles ARIA **non disponibles en HTML5** / composants personnalisés

SVG

- Format d'image **vectoriel**
 - Fonctionne comme HTML mais avec des primitives graphiques
 - Modification du style avec CSS (et quelques attributs spécifiques)
 - Peut être manipulé via le DOM (cf. plus tard...)
- Balise racine `<svg>`
- Regroupement de différents éléments à l'aide de balises `<g>`, équivalent d'un `<div>`
- Références des différents éléments SVG disponible sur [MDN](#)

SVG

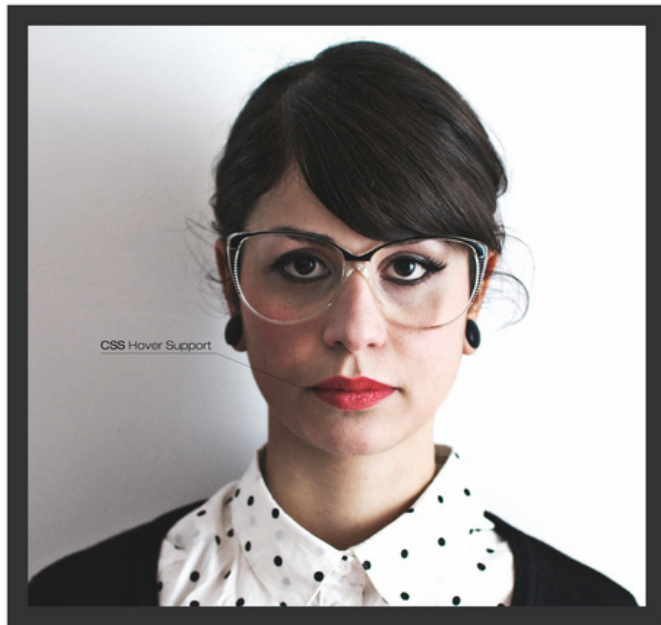
```
<svg viewBox="0 0 300 200">  
  <rect width="100%" height="100%" fill="red"/>  
  <circle cx="150" cy="2100" r="80" fill="green"/>  
  <text x="150" y="125" font-size="60" text-anchor="middle"  
fill="white">SVG</text>  
</svg>
```



Fin du HTML et SVG

Passons au CSS !

CSS Mess



any normal browser



internet explorer 6